МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЁТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Рекурсивная обработка иерархических списков

Студент гр. 8304		Бочаров Ф.Д.
Преподаватель		Фирсов М.А.

Санкт-Петербург

2019

Задание.

Вариант №8

Заменить в иерархическом списке все вхождения заданного элемента (атома) x на заданный элемент (атом) y;

Цель работы.

Решить полученную задачу, используя иерархические списки. Получение навыков работы с нелинейными структурами данных.

Описание алгоритма.

Сначала принимаемое значение записываем в строку. После чего проверяет ее на корректность и, если строка верна, создает список при помощи функций считывания данных. Если ввод выполнен верно, программа просит ввести элемент для замены и новый элемент. Если элементы отличаются, то функцией swapElements заменяется один элемент на другой. Программа выводит новый список.

Описание функций программы:

1) `static void swapElements(ListPointer list, const char oldElem, const char newElem);`

   Рекурсивная функция замены одного элемента на другой. Принимает указатель на список, элемент который надо заменить и элемент на который надо заменить.

2) `HierarchicalList::buildList(ListPointer &list, const std::string& str)`

   Функция создания иерархического списка. Принимает на вход строку, проверяет её на корректность и вызывает метод readData().

3)`readData (ListPointer& list, const char prev, std::string::const_iterator& it, const std::string::const_iterator& end);`

Функция считывания данных. Принимает указатели на список, начало и конец строки. строки.

3) `readSeq(ListPointer&list,std::string::const_iterator&it, const std::string::const_iterator& end)`

Функция считывания списка. Принимает указатели на список, начло и конец строки.

Выводы.

В ходе выполнения работы были изучены нелинейные структуры данных, был получен опыт работы с иерархическими списками. Получены навыки реализации

Протокол

Тестирование:

Входные данные	Выходные данные
(a) a b	(b)
(a(b(c))) c d	(a(b(d)))
(af(f(s))) f s	(af(s(s)))
(a(v(c(s(f(h(w))))))) h d	(a(v(c(s(f(d(w)))))))
1()	Wrong Input
()wd)w	Wrong Input
Rw()dd	Wrong Input

Исходный код

hierarchicallist.cpp

```
#include "hierarchicallist.h"

HierarchicalList::HierarchicalList()
{
    /*
     * По умолчанию объект класса является пустым списком
    */

    flag = false;
    atom = 0;

}


HierarchicalList::~HierarchicalList()
{

}


bool HierarchicalList::isNull() const
{
    /*
     * Возвращает true, если элемент является нулевым списком,
     * false - в ином случае
    */

    return (!flag && nestedList == nullptr && nextList == nullptr);
}


HierarchicalList::ListPointer HierarchicalList::getHead() const
{
    /*
     * Если элемент не атом - возращает указатель на вложенный список,
     * в ином случае - nullptr
    */

    if (!flag) {
        return nestedList;
    }
    else {
        std::cerr << "Error: Head(atom)\n";
        return nullptr;
    }
}


HierarchicalList::ListPointer HierarchicalList::getTail() const
{
    /*
     * Если элемент не атом - возращает указатель на следующий список,
     * в ином случае - nullptr
    */

    if (!flag) {
        return nextList;
    }
    else {
```

```
        std::cerr << "Error: Tail(atom)\n";
        return nullptr;
    }
}


bool HierarchicalList::isAtom() const
{
    /*
     * Если элемент атом - возращает true,
     * в ином случае - false
    */

    return flag;
}


HierarchicalList::ListPointer HierarchicalList::cons(ListPointer& head,
ListPointer& tail)
{
    /*
     * Функция создания списка
    */

    if (tail != nullptr && tail->isAtom()) {
        std::cerr << "Error: Tail(atom)\n";
        return nullptr;
    }
    else {
        ListPointer tmp(new HierarchicalList);
        tmp->nestedList = head;
        tmp->nextList = tail;
        return tmp;
    }
}


void HierarchicalList::print_seq(std::ostream& out) const
{
    /*
     * Функция печати Tail
    */

    if (!isNull()) {
        out << this->getHead();

        if (this->getTail() != nullptr)
            this->getTail()->print_seq(out);
    }
}


void HierarchicalList::swapElements(ListPointer list, const char oldElem,
const char newElem, size_t depth)
{
    /*
     * Рекурсивная функция замены одного элемента списка на другой.
     * Принимает на вход умный указатель на список, элемент, который надо за-
менить и
     * элемент, на который надо заменить. Глубина рекурсии передается пара-
метром depth.
     *
     * Если текущий список "пустой" - это базовый случай.
     * Если текущий список "не атом" - это рекурсивный случай.
```

```
     *
     * Если список - атом, проверяется, совпадает ли элемент с тем, который
надо заменить,
     * и происходит замена.
    */


    //Если список пустой
    if (list == nullptr || list->isNull()) {
        return;
    }
    //Если список - атом
    else if (list->isAtom()) {
        if (list->atom == oldElem) {
            list->atom = newElem;
        }
    }
    //Вызов рекурсии для следующих списков
    else {
        swapElements(list->nestedList, oldElem, newElem, depth);
        swapElements(list->nextList, oldElem, newElem, depth + 1);
    }

}


std::ostream& operator<<(std::ostream& out, const Hierarchical-
List::ListPointer& list)
{
    /*
     * Перегрузка оператора вывода
    */

    if (list == nullptr || list->isNull()) {
        out << "()";
    }
    else if (list->isAtom()) {
        out << list->getAtom();
    }
    else {
        out << "(";

        out << list->getHead();
        if (list->getTail() != nullptr)
            list->getTail()->print_seq(out);

        out << ")";
    }

    return out;
}


bool HierarchicalList::isCorrectStr(const std::string& str)
{
    /*
     * Функция проверки корректности входных данных,
     * принимает на вход ссылку на строку, проверяет размер и структуру
строки,
     * возвращает true, если строка корректна, и false в ином случае
    */

    std::cout << "isCorrect str: " << str << "\n";
    int countBracket = 0;
```

```
    if (str.size() < 2)
        return false;

    if (str[0] != '(' || str[str.size() - 1] != ')')
        return false;

    size_t i;
    for (i = 0; i < str.size(); ++i) {
        char elem = str[i];
        if (elem == '(')
            ++countBracket;
        else if (elem == ')')
            --countBracket;
        else if (!isalpha(elem))
            return false;

        if (countBracket <= 0 && i != str.size()-1)
            return false;
    }

    if (countBracket > 0 || i != str.size()) {
        return false;
    }

    std::cout << "Str is correct.\n";
    return true;
}


void HierarchicalList::createAtom(const char ch)
{
    /*
     * Создается объект класса - атом
    */
    this->atom = ch;
    this->flag = true;
}


void HierarchicalList::readData(ListPointer& list, const char prev,
std::string::const_iterator &it,
                                const std::string::const_iterator& end)
{
    /*
     * Функция считывания данных. Считывает либо атом, либо рекурсивно считы-
вает список
    */

    if (prev != '(') {
        list->createAtom(prev);
    }
    else {
        readSeq(list, it, end);
    }
}


void HierarchicalList::readSeq(ListPointer& list, std::string::const_itera-
tor&it,
                               const std::string::const_iterator& end)
{
    /*
     * Функция считывания списка. Рекусривно считывает данные и спискок и
```

```
     * добавляет их в исходный.
    */

    ListPointer headList(new HierarchicalList);
    ListPointer tailList(new HierarchicalList);

    if (it == end)
        return;

    if (*it == ')') {
        ++it;
    }
    else {
        char prev = *it;
        ++it;
        readData(headList, prev, it, end);
        readSeq(tailList, it, end);
        list = cons(headList, tailList);
    }
}


bool HierarchicalList::buildList(ListPointer& list, const std::string& str)
{
    /*
     * Функция создания иерархического списка. Принимает на вход ссылку
     * на строку, проверяет корректность строки и вызывает приватный метод
     * readData().
    */

    std::cout << "Creating list from str: " << str << "\n";

    if (!isCorrectStr(str)) {
        std::cout << "Str is incorrect!!!\n";
        return false;
    }

    auto it_begin = str.cbegin();
    auto it_end = str.cend();
    char prev = *it_begin;
    ++it_begin;

    readData(list, prev, it_begin, it_end);

    return true;
}


char HierarchicalList::getAtom() const
{
    /*
     * Функция возвращает значение атома
    */
    if (flag) {
        return atom;
    }
    else {
        std::cerr << "Error: getAtom(!atom)\n";
        return 0;
    }
}
```

hierarchicallist.h

```
#ifndef HIERARCHICALLIST_H
#define HIERARCHICALLIST_H

#include <memory>
#include <iostream>


class HierarchicalList
{
public:
    //Умный укаазатель на список для избежания утечек памяти
    typedef std::shared_ptr<HierarchicalList> ListPointer;

    explicit HierarchicalList();
    ~HierarchicalList();

    //Функции для доступа к данным, head - указатель на вложенные списки,
tail - на следующие
    ListPointer getHead() const;
    ListPointer getTail() const;
    bool isNull() const;
    bool isAtom() const;
    char getAtom() const;

    //static-методы можно вызывать как обычные функции, не создавая объект
класса
    static bool buildList(ListPointer& list, const std::string& str);

    //Перегрузка оператора cout для печати списка, и функция рекурсивной пе-
чати следующих списков
    friend std::ostream& operator<< (std::ostream& out, const ListPointer&
list);
    void print_seq(std::ostream& out) const;

    //Функция замены одного элемента на другой
    static void swapElements(ListPointer list, const char oldElem, const char
newElem, size_t depth = 1);

private:
    //Функция проверки входной строки на корректность
    static bool isCorrectStr(const std::string& str);

    //функции рекурсивного чтения списка
    static void readData(ListPointer& list, const char prev,
std::string::const_iterator& it,
                         const std::string::const_iterator& end);
    static void readSeq(ListPointer& list, std::string::const_iterator& it,
                        const std::string::const_iterator& end);

    //Функция конструирования непустого списка
    static ListPointer cons(ListPointer& head, ListPointer& tail);

    //превращает объект класса в атом(список с данными)
    void createAtom(const char ch);


private:
    //Флаг для определения, является ли список - атомом
    bool flag;

        //указатели на вложенный и следующий список
```

```
    ListPointer nestedList;
    ListPointer nextList;
    //данные
        char atom;

};

#endif // HIERARCHICALLIST_H
```

main.cpp

```
#include "hierarchicallist.h"
#include <iostream>
#include <fstream>
#include <string>


int main(int argc, char *argv[]) {
    std::string listStr;
    int choice;
    HierarchicalList::ListPointer list(new HierarchicalList);
    char oldElem, newElem;

    while (true) {
        std::cout << "What type of test?" << std::endl;
        std::cout << "1.FROM FILE" << std::endl;
        std::cout << "2.FROM CONSOLE" << std::endl;
        std::cin >> choice;

          switch (choice){
              case 1:{
                  std::cout << " FROM FILE " << std::endl;
                  std::string fileName = "./Tests/";
                   fileName += argv[1];
                  std::ifstream inputFile(fileName, std::ios::in);
                  inputFile >> listStr >> oldElem >> newElem;
                  HierarchicalList::buildList(list, listStr);

                      if (oldElem != newElem) {
                        HierarchicalList::swapElements(list, oldElem, new-
Elem);
                        std::cout << "List after swap:" << list << "\n";
                      }
                      else {
                        std::cout << "Elements are equal. List hadn't
changed\n";
                      }
                  inputFile.close();
                  break;
              }

            case 2:
              std::cout << "Enter list: ";
              std::cin >> listStr;
              std::cout << "List : " << listStr << "\n";


              if (HierarchicalList::buildList(list, listStr)) {

                std::cout << "Enter old element:";
                std::cin >> oldElem;

                std::cout << "Enter new element:";
```

```
                std::cin >> newElem;


                if (oldElem != newElem) {
                  HierarchicalList::swapElements(list, oldElem, newElem);
                  std::cout << "List after swap:" << list << "\n";
                }
                else {
                  std::cout << "Elements are equal. List hadn't changed\n";
                }

          }
                    break;

        }

    }
    return 0;
}
```